DOUGLAS A. FEMEC

MANUFACTURING-TEST ENGINEERING EXPERT-WITNESS REPORT

ECIMOS, LLC. v CARRIER CORPORATION
CASE NO. 2:15-cv-02726-JPM-cgc

TABLE OF CONTENTS

MANUFACTURING-TEST ENGINEERING EXPERT-WITNESS REPORT

ECIMOS, LLC. v CARRIER CORPORATION, CASE NO. 2:15-cv-02726-JPM-cgc

INTRODUCTION

This report contains expert-witness opinions from a senior, practicing manufacturing-test engineer and relational-database designer on the matter of ECIMOS, LLC. v Carrier Corporation, Case No. 2:15-cv-02726-JPM-cgc. In specific, these opinions will focus on the case's copyright and intellectual-property (IP) issues from the perspective of manufacturing testing. Comparisons will be made between the individual applications and between their associated databases.

The materials relied upon in generating this report include

- The source code for ECIMOS's ECI application;
- The schema for the database associated with ECIMOS's ECI application, including all database-related objects such as tables and stored procedures;
- The source code for Carrier's RES application;
- The schema for the database associated with Carrier's RES application, including, as above, all database-related objects such as tables and stored procedures; and
- James M. Chenault's expert-witness report, "JMC ECIMOS vs Carrier Report (Exhibit A)."

A visit to Carrier's manufacturing plant, located in Collierville, TN, provided observations of air-conditioner testing being performed using Carrier's RES application.

In addition to the above materials and visit, the expert-witness opinions contained herein are based on 25+ years of experience in the design and implementation of relational databases in laboratory- and manufacturing-testing environments, in architecting and developing applications for these same environments using Microsoft® Visual Basic® and National Instruments™ LabVIEW™ and TestStand, and in a variety of industries through full-time positions and consultancies. This expert witness has earned distinctions in software development as both a Certified LabVIEW™ Architect and a Certified TestStand Developer (awarded by National Instruments™).

REGARDING SOFTWARE ARCHITECTURES

For ease of understanding, rather than using the generic manufacturing-test terms of unit and UUT (unit-under-test), the following will use "air conditioner," since, in Carrier's case, a single air conditioner would correspond to a single UUT.

The software architectures utilized in both ECIMOS's ECI application and Carrier's RES application lend themselves well to the Sixth Circuit's preferred Abstraction-Filtration-Comparison (AFC) test. The source code for both of these applications is compartmentalized and, hence, can be abstracted into portions that

- Interact with the user;
- Identify the air conditioner being tested and obtained the respective adjustable test parameter(s) from the database;
- Perform the tests specified in the database, including communicating with the air conditioner and the appropriate measurement instruments; and
- Record the test results in the database.

When it comes to applying filtration to the code bases (i.e., the entire collections of source code used to write the applications) for these two applications, it is important to keep in mind that any "communication" between an application and a third-party product is dictated by that third-party product in a form freely available to any authorized user of that product. As a result, the "communication" schemes employed by the individual applications, whether for communicating with a relational database or a piece of commercial, off-the-shelf (COTS) hardware, are removed as non-protectable elements prior to any comparisons being made.

The only areas left for comparison, excluding those differences that arise simply from the differing mechanics of the two languages (Microsoft® Visual Basic® and National Instruments™ LabVIEW™), involve the determinations of whether a given test has passed. Neither code base makes use of elaborate data-analysis schemes, but rather they both simply determine whether a measured value falls within a specified range of values.

In short, application of the AFC test finds no evidence of copyright infringement or "pirated" software in comparing the source code for Carrier's RES application to the source code of ECIMOS's ECI application. Neither application contains potentially protectable content along the lines of elaborate data analysis algorithms.

REGARDING RES POSSIBLY ACCESSING ECI

The contention that Carrier's RES application makes use of all or some portion of ECIMOS's ECI application can be countered using the following points:

- LabVIEW™ is largely not a text-based language; it can be described as a node-based language, where nodes are connected via "wires" in a graphical environment. "Largely" is used here because there are nodes, specifically structures in this case, into which formulas can be entered using text rather than LabVIEW™'s mathematical nodes. Formula nodes, for instance, accept code akin to C, while MATLAB® script nodes accept MATLAB® code. There are no nodes, though, that accept Microsoft® Visual Basic® code, whether from Visual Basic® 6 (VB6) or Visual Basic® .NET (VB.NET). Further, Carrier's RES application makes use of neither formula nodes nor MATLAB® script nodes, as can be demonstrated by searching the application's source code for these nodes, starting at the project level.

- Executables (i.e., files with a .exe extension) are the final result of many application "build" processes, such as that for ECIMOS's ECI application. Source code is compiled to create object files; executables are built from object files. LabVIEW™ is capable of starting (or "calling") executables, but it is not capable of "driving" the user interface of a non- LabVIEW™ executable. Further, executables are started in LabVIEW™ via System Exec.vi, a "function" provided by National Instruments™ as part of LabVIEW™. A search of the source code for Carrier's RES application found that System Exec.vi is used, but not for calling ECIMOS's ECI application; instead, it is used to start the Windows text editor Notepad, to present some persistent dialog boxes (such as for test failures), and to perform some operations against the associated database.

- In addition to executables, dynamic link libraries (DLLs) can also be built from object files. The source code that compiles to give the object files must be specifically written to make its functions available (to "expose" its functions) through a DLL. The VB source code for ECIMOS's ECI application was not written to make available its functions through a DLL. Further, the LabVIEW™ Call Library Function Node, which LabVIEW™ would use to access a DLL's exposed functions, is not found in the source code for Carrier's RES application.

- In the .NET world, the .dll extension, previously used to denote a dynamic link library, is used for a new kind of file, namely assemblies. While the DLLs described above can contain single functions that are exposed for access by other applications, an assembly makes available the constructor and the public methods and properties of an object. For LabVIEW™, or any other application, to make use of an assembly, it needs to first call the constructor contained in the assembly to instantiate a member of the assembly's class. LabVIEW™ would do this via the Constructor Node, but this node is not found in the source code for Carrier's RES application.

REGARDING DATABASE STRUCTURES

Both ECIMOS's ECI application and Carrier's RES application make use of relational databases. As will be presented below at a primer level, relational databases are composed of multiple tables. Groups of tables within a single database are related to each other – hence the modifier "relational" – through the use of minimal sets of common information.

PRIMER ON RELATIONAL DATABASES

Consider, for instance, how a relational database would be used by an on-line ordering site. Each customer order needs to be uniquely identified. Each item ordered is typically identified by a part or model number. A simple, but non-relational, database for this scenario would contain a single table. A single row in that table would describe all items requested in a single customer order. Now consider in some detail what information needs to go into each row of that table.

To properly bill the customer, each row would need to contain billing information such as the number of the credit card to be used, the credit card's security code, the complete name on the credit card, and the billing address for the account associated with the credit card. To properly deliver the items ordered, each row would need to contain shipping information such as the customer's address, the preferred method of shipping, and contact information such as phone numbers and e-mail addresses. To properly identify the items ordered, each row would need to contain the part number and the quantity ordered for each item ordered. This last requirement causes a problem, or, rather, it is a stumbling block for less-experienced database developers.

An individual table in a database can be viewed as a ledger one would keep on paper. A ledger has columns for each piece of information to be entered. The columns in a ledger are determined in advance of the ledger being used, the number of columns is fixed (although columns can be added or removed later), and there should not be too many columns that are not used every time a row on the ledger is filled in. This is where the last requirement shows itself to be a problem. The on-line ordering site does not want to limit how many items a customer can order, but without limiting the number, the site's database developers do not know how many columns to put into the table for specifying the items ordered.

Having only one pair of columns – part number and quantity – would limit each order to only one item. Having one hundred pairs of columns would allow the ordering of multiple items, but all one-hundred pairs of columns would not be used for every order, potentially resulting in a lot of blank space. Lots of blank space, whether on a paper ledger or a database located on a computer's hard drive, is wasteful and, potentially, expensive. (And, no, all of the columns necessary to describe a single item to be ordered have not been considered here; adding columns for size and color, for instance, just makes the problem worse.)

Returning to the ledger analogy, consider what was typically done in the days when general stores maintained ledgers of customer purchases:

CUSTOMER	ACCOUNT	DATE	ITEM	QUANTITY	COST	PAID
Mrs. Johnson	Edgar Johnson	Jan 8th	Seed potatoes	250 lbs.	$25.00	
"	"	"	Seed corn	12 bushels	$48.00	
"	"	"	Flannel cloth	1 bolt	$8.00	
				Total	$81.00	$81.00
Mr. Frederick	Benjamin Frederick	Jan 8th	Shovel	1	$3.50	$3.50
Mr. Johnson	Edgar Johnson	Jan 9th	Shotgun shells	3 boxes of 24	$3.60	
"	"	"	Bear trap	1	$3.00	
				Total	$6.60	$6.60
…	…	…	…	…	…	

This table depicts the scenario where a single row represents the purchase of whatever quantity of a single item. Note the use of ditto marks (“) to denote information repeated from above. This dittoed information is a clue to how this ledger could be transformed into a relational database; dittoed information should be stored once for each order and then referenced to each item ordered.

A typical relational database for this scenario, then, would contain a table that lists all of the customer orders, including mailing addresses and billing information but not the specific items ordered. A unique identifier would be generated for each entry or row in this table. A second table would list all of the items ordered, along with any information related to each item ordered such as quantity, size, and color; this second table would also contain the unique identifier generated for each customer order, which is used to find all items associated with a single customer order.

RELATIONAL DATABASES IN MANUFACTURING TESTING

Relational databases for storing results from manufacturing tests can be similarly composed of two types of tables, one for keeping track of the particular air conditioner tested and another for recording the individual test results for each air conditioner. The first table, the one that is used to keep track of the individual air conditioners being tested, could look something like this:

TABLE ENTRY NUMBER FOR THIS TIME THIS AIR CONDITIONER WAS TESTED	MODEL OF THE AIR CONDITIONER THAT WAS TESTED	SERIAL NUMBER OF THE AIR CONDITIONER THAT WAS TESTED	DATE AND TIME THE TESTING WAS PERFORMED	OVERALL TEST RESULT (PASS OR FAIL)	...
...	...	...	...	...	...
26	ACN034	320908357095	October 16, 2017 2:15:33 P.M.	PASS	...
27	ACN044	002347724387	October 16, 2017 2:28:02 P.M.	FAIL	...
28	ACN044	002347724387	October 16, 2017 2:45:58 P.M.	PASS	...
...	...	...	...	...	...

The information contained in this table, and in the one that follows, is solely for purposes of illustration and is not meant to represent actual model numbers, serial numbers, or test results. The ellipses in the final column and in the first and last rows of the displayed portions of these tables are meant to indicate that there can be additional columns and rows in this table; only the columns and rows necessary for this illustration have been shown.

The design of this table allows for an individual air conditioner to be tested multiple times. In the case of the model ACN044 air conditioner with serial number 002347724387, one or more individual tests failed the first time it was tested. That is why the table has a FAIL value in the OVERALL TEST RESULT (PASS OR FAIL) column for when that air conditioner was tested on October 16, 2017, at 2:28:02 P.M. After briefly troubleshooting the problem, the operator was able to run the test again, and this time – on October 16, 2017, at 2:45:58 P.M. – the air conditioner passed all of the tests.

The second table, the one that contains the individual test results, could look something like this:

TABLE ENTRY NUMBER FOR THIS TIME THIS AIR CONDITIONER WAS TESTED USING THIS SPECIFIC TEST	TABLE ENTRY NUMBER FOR THIS TIME THIS AIR CONDITIONER WAS TESTED	NAME OF THE SPECIFIC TEST BEING PERFORMED	DATE AND TIME THIS SPECIFIC TEST WAS PERFORMED	RESULT OF THIS SPECIFIC TEST (PASS OR FAIL)	…
…	…	…	…	…	…
213	27	Power On	October 16, 2017 2:28:03 P.M.	PASS	…
214	27	Idle Current Check	October 16, 2017 2:28:07 P.M.	PASS	…
215	27	Set Hipot to AC Mode	October 16, 2017 2:28:11 P.M.	PASS	…
216	27	AC Current Check	October 16, 2017 2:28:23 P.M.	FAIL	…
217	27	Power Off	October 16, 2017 2:28:25 P.M.	PASS	…
…	…	…	…	…	…

Note the shaded column heading in both of these tables, and that the shaded wording is identical. These columns provide a link between the two tables, allowing one to see which tests were performed on each air conditioner each time it was tested. This link also keeps all of the information in each row of the first table from being repeated in the respective rows of the second table. This is an identifying feature of a relational database – links that provide relationships between tables.

These links make use of what are called "keys" in relational-database terminology. The primary key of a table identifies the column (or combination of columns) that uniquely identifies each row in the table. Because primary keys must uniquely identify each row, any primary-key value can appear only once in the table. In the case of the first table, a combination of MODEL OF THE AIR CONDITIONER THAT WAS TESTED (when serial numbers are not unique across all model numbers), SERIAL NUMBER OF THE AIR CONDITIONER THAT WAS TESTED, and DATE AND TIME THE TESTING WAS PERFORMED should uniquely identify each row in the table. A simpler primary key for this table – the one shaded above – would be TABLE ENTRY NUMBER FOR THIS TIME THIS AIR CONDITIONER WAS TESTED; why this would be a simpler key should become apparent presently.

For any single row in the first table, there can be multiple rows in the second table. The "mapping" between these two tables makes use of the first table's primary key. The column in the second table with the same heading – the one shaded above – is referred to as a foreign key (i.e., it is not a key the values for which come from the content of the other columns in the second table, hence it is "foreign" to the second table). Since multiple individual tests can be executed as part of testing an air conditioner, any single value found in the column TABLE ENTRY NUMBER FOR THIS TIME THIS AIR CONDITIONER WAS TESTED in the first table can be found any number of times in the column with the same heading in the second table. This is referred to as a one-to-many relationship, sometimes written as $1 \rightarrow \infty$.

That the foreign key of the second table needs to contain the values of the primary key of the first table is why the single column TABLE ENTRY NUMBER FOR THIS TIME THIS AIR CONDITIONER WAS TESTED is a simpler primary key for the first table. Otherwise, the second table would have required three additional columns, namely MODEL OF THE AIR CONDITIONER THAT WAS TESTED, SERIAL NUMBER OF THE AIR CONDITIONER THAT WAS TESTED, and DATE AND TIME THE TESTING WAS PERFORMED, rather than just one. Note that the second table has its own primary key, namely the column TABLE ENTRY NUMBER FOR THIS TIME THIS AIR CONDITIONER WAS TESTED WITH THIS SPECIFIC TEST.

The column headings in the above sample tables are much longer than would typically be used in a database table; in the latest versions of Microsoft® SQL Server, for instance, these column headings, which would be referred to as column or field names, can be composed of no more than 128 letters and numbers; only a few special characters are allowed in these names. Because of these constraints, common short versions of these names are routinely used. For instance, instead of SERIAL NUMBER OF THE AIR CONDITIONER THAT WAS TESTED, shorter versions such as SERIALNUMBER, SERIAL_NO, or SN could be used; similarly, for MODEL OF THE AIR CONDITIONER THAT WAS TESTED, MODEL, MODEL_NO, or MN could be used. As a result, the above two tables would more likely look like the following:

FULLTESTID	**MODEL**	**SN**	**TESTDATETIME**	**OVERALLRESULT**	**...**
...	...	...	...	...	...
26	ACN034	320908357095	October 16, 2017 2:15:33 P.M.	PASS	...
27	ACN044	002347724387	October 16, 2017 2:28:02 P.M.	FAIL	...
28	ACN044	002347724387	October 16, 2017 2:45:58 P.M.	PASS	...
...	...	...	...	...	...

TESTID	**FULLTESTID**	**TESTNAME**	**TESTDATETIME**	**TESTRESULT**	**...**
...	...	...	...	...	...
213	27	Power On	October 16, 2017 2:28:03 P.M.	PASS	...
214	27	Idle Current Check	October 16, 2017 2:28:07 P.M.	PASS	...
215	27	Set Hipot to AC Mode	October 16, 2017 2:28:11 P.M.	PASS	...
216	27	AC Current Check	October 16, 2017 2:28:23 P.M.	FAIL	...
217	27	Power Off	October 16, 2017 2:28:25 P.M.	PASS	...
...	...	...	...	...	...

It is worth noting that each column or field in a database table has a defined data type. Commonly used data types include integers, fractional numbers, date-and-time values, and strings. In these tables, the likely data types are as follows:

First table: FULLTESTID is an integer (the primary key for this table),

MODEL is a string (to allow letters, numbers, and a few special characters to be used in specifying it),

SN is a string (to allow letters, numbers, and a few special characters to be used in specifying it),

TESTDATETIME is a date-and-time value, and

OVERALLRESULT is a string.

Second table: TESTID is an integer (the primary key for this table),

FULLTESTID is an integer (must be the same data type used for FULLTESTID in the first table, since this is a foreign key),

TESTNAME is a string,

TESTDATETIME is a date-time value, and

TESTRESULT is a string.

These points regarding how relational databases are constructed should prove useful in understanding the following comparisons between the database used by ECIMOS's ECI application and the database used by Carrier's RES application.

COMPARING THE ECIMOS ECI AND CARRIER RES DATABASES

Because of the number of columns found in the database tables used by ECIMOS's ECI application and Carrier's RES application, tabular depictions of these databases will now be turned on their sides and no sample values will be included.

The databases for ECIMOS's ECI application and for Carrier's RES application both contain tables for detailing the tests to be performed and for storing test results. The largest architectural difference between the two databases is found in the tables used for storing results, and this difference is readily understood from the primer presented earlier. (The tables used for storing testing details are "structurally" nearly identical, yet they will be discussed later.)

In short, the database associated with Carrier's RES application contains three tables for storing test results: the first for recording each instance of an air conditioner being tested, including identifying information for the particular air conditioner, the second for recording individual test values, and the third for recording whether individual tests passed. This design, described under Relation Database in Manufacturing Testing, is consistent with current best practices (although the second and third tables could be combined into a single table). Here is a depiction of the table named RT_RunTest_Main, for recording each instance of an air conditioner being tested:

COLUMN NAME	DATA TYPE	KEY TYPE
ID	Integer	Primary Key
DateRun	Date-Time	
Serial	String	
Model	String	
TestNo	Integer	
PassFail	String	
ErrorMessage	String	
TestDescription	String	
Operator	String	
Line	String	
Station	String	
Shift	Integer	
RTVersion	String	
UnitType	String	
CycleTime	Fractional Number	

Here is a depiction of the table named RT_RunTest_Parameter_Detail, in which individual test values are stored:

COLUMN NAME	DATA TYPE	KEY TYPE
ID	Integer	Primary Key
SeqStepNo	Integer	Primary Key
Script	String	Primary Key
RunNo	Integer	Primary Key
StepNo	Integer	Primary Key
Parameter	String	Primary Key
ActualValue	Fractional Number	
HiLimit	Fractional Number	
LoLimit	Fractional Number	
StrValue	String	
StrLimit	String	

In the database associated with Carrier's RES Application, there is one of these tables of individual test values for each parameter. Adding additional parameters amounts to creating additional copies of this single table. A single table, design in the same way as this one, could also have been used; in this case, no additional tables would need to be created as new parameters are added. Parameters to be removed are simply no longer written to this table; the data should be retained for historical purposes, but entries for these removed parameters will no longer be added to the database with future testing.

Here is a depiction of the table RT_RunTest_Test_Detail, in which whether individual tests passed is stored:

COLUMN NAME	**DATA TYPE**	**KEY TYPE**
ID	Integer	Primary Key
StepNo	Integer	Primary Key
Script	String	Primary Key
PassFail	String	
CycleTime	Fractional Number	

The ECIMOS's ECI application, on the other hand, makes use of a single table for storing test results for and descriptive information about the particular air conditioner being tested. Referring to Primer on Relational Databases, the design of this table best fits the worst-case scenario of handling a maximum number of one hundred items per order by putting one hundred pairs of columns into the table (or, worse, having a column for every item that can be ordered):

COLUMN NAME	**DATA TYPE**	**KEY TYPE**
Serial	String	Primary Key
ModelRun	String	
ErrorMessage	String	
TestDescription	String	
DateRun	Date-Time	Primary Key
Operator	String	
Line	String	
Station	String	
Shift	Integer	
TestNo	Fractional Number	
CycleTime	Integer	
PassFail	String	
RefrigerantType	String	
ODFSailSwitch	Integer	
DefrostCycleOK	Integer	
RunVolts	Fractional Number	
HipotVolts	Fractional Number	
HipotLeakage	Fractional Number	
HipotString	Fractional Number	
InsulationResVolts	Fractional Number	
InsulationResLeakage	Fractional Number	
GroundBondAmps	Fractional Number	
GroundBondResistance	Fractional Number	
AmbientTemp	Fractional Number	
LiquidTempCooling	Fractional Number	

COLUMN NAME	DATA TYPE	KEY TYPE
LiquidTempHeating	Fractional Number	
StartVoltsLoParam	Fractional Number	
StartVoltsActual	Fractional Number	
StartVoltsHiParam	Fractional Number	
StartAmpsLoParam	Fractional Number	
StartAmpsActual	Fractional Number	
StartAmpsHiParam	Fractional Number	
PeakAmpsActual	Fractional Number	
L1CurrentCoolingLoParam	Fractional Number	
L1CurrentCoolingActual	Fractional Number	
L1CurrentCoolingHiParam	Fractional Number	
LiquidPressureCoolingLoParam	Fractional Number	
LiquidPressureCoolingActual	Fractional Number	
LiquidPressureCoolingHiParam	Fractional Number	
VaporPressureCoolingLoParam	Fractional Number	
VaporPressureCoolingActual	Fractional Number	
VaporPressureCoolingHiParam	Fractional Number	
VibrationCoolingLoParam	Fractional Number	
VibrationCoolingActual	Fractional Number	
VibrationCoolingHiParam	Fractional Number	
DeltaTemperatureCoolingLoParam	Fractional Number	
DeltaTemperatureCoolingActual	Fractional Number	
DeltaTemperatureCoolingHiParam	Fractional Number	
L1CurrentHeatingLoParam	Fractional Number	
L1CurrentHeatingActual	Fractional Number	
L1CurrentHeatingHiParam	Fractional Number	
LiquidPressureHeatingLoParam	Fractional Number	
LiquidPressureHeatingActual	Fractional Number	
LiquidPressureHeatingHiParam	Fractional Number	
VaporPressureHeatingLoParam	Fractional Number	
VaporPressureHeatingActual	Fractional Number	
VaporPressureHeatingHiParam	Fractional Number	
VibrationHeatingLoParam	Fractional Number	
VibrationHeatingActual	Fractional Number	
VibrationHeatingHiParam	Fractional Number	
DeltaTemperatureHeatingLoParam	Fractional Number	
DeltaTemperatureHeatingActual	Fractional Number	
DeltaTemperatureHeatingHiParam	Fractional Number	
L1CurrentLowCoolingLoParam	Fractional Number	
L1CurrentLowCoolingActual	Fractional Number	
L1CurrentLowCoolingHiParam	Fractional Number	
LiquidPressureLowCoolingLoParam	Fractional Number	
LiquidPressureLowCoolingActual	Fractional Number	
LiquidPressureLowCoolingHiParam	Fractional Number	
VaporPressureLowCoolingLoParam	Fractional Number	
VaporPressureLowCoolingActual	Fractional Number	

COLUMN NAME	DATA TYPE	KEY TYPE
VaporPressureLowCoolingHiParam	Fractional Number	
L1CurrentLowHeatingLoParam	Fractional Number	
L1CurrentLowHeatingActual	Fractional Number	
L1CurrentLowHeatingHiParam	Fractional Number	
LiquidPressureLowHeatingLoParam	Fractional Number	
LiquidPressureLowHeatingActual	Fractional Number	
LiquidPressureLowHeatingHiParam	Fractional Number	
VaporPressureLowHeatingLoParam	Fractional Number	
VaporPressureLowHeatingActual	Fractional Number	
VaporPressureLowHeatingHiParam	Fractional Number	
StartWindingCurrentLoParam	Fractional Number	
StartWindingCurrentActual	Fractional Number	
StartWindingCurrentHiParam	Fractional Number	
FANCurrentLoParam	Fractional Number	
FANCurrentActual	Fractional Number	
FANCurrentHiParam	Fractional Number	
FANLowCurrentLoParam	Fractional Number	
FANLowCurrentActual	Fractional Number	
FANLowCurrentHiParam	Fractional Number	
CCHCurrentLoParam	Fractional Number	
CCHCurrentActual	Fractional Number	
CCHCurrentHiParam	Fractional Number	
MoistureLoParam	Fractional Number	
MoistureActual	Fractional Number	
MoistureHiParam	Fractional Number	
L1Current24VACLoParam	Fractional Number	
L1Current24VACActual	Fractional Number	
L1Current24VACHiParam	Fractional Number	
StartWindingCurrent24VACLoParam	Fractional Number	
StartWindingCurrent24VACActual	Fractional Number	
StartWindingCurrent24VACHiParam	Fractional Number	
FANCurrentSteadyStateLoParam	Fractional Number	
FANCurrentSteadyStateActual	Fractional Number	
FANCurrentSteadyStateHiParam	Fractional Number	
L1CurrentSteadyStateLoParam	Fractional Number	
L1CurrentSteadyStateActual	Fractional Number	
L1CurrentSteadyStateHiParam	Fractional Number	
StartWindingCurrentSteadyStateLoParam	Fractional Number	
StartWindingCurrentSteadyStateActual	Fractional Number	
StartWindingCurrentSteadyStateHiParam	Fractional Number	
StartWindingLowCurrentLoParam	Fractional Number	
StartWindingLowCurrentActual	Fractional Number	
StartWindingLowCurrentHiParam	Fractional Number	
DeltaTemperatureLowCoolingLoParam	Fractional Number	
DeltaTemperatureLowCoolingActual	Fractional Number	
DeltaTemperatureLowCoolingHiParam	Fractional Number	

COLUMN NAME	DATA TYPE	KEY TYPE
DeltaTemperatureLowHeatingLoParam	Fractional Number	
DeltaTemperatureLowHeatingActual	Fractional Number	
DeltaTemperatureLowHeatingHiParam	Fractional Number	
EvapLiquidPressLoParam	Fractional Number	
EvapLiquidPressActual	Fractional Number	
EvapLiquidPressHiParam	Fractional Number	
EvapVaporPressLoParam	Fractional Number	
EvapVaporPressActual	Fractional Number	
EvapVaporPressHiParam	Fractional Number	
TXVPressCoolingLoParam	Fractional Number	
TXVPressCoolingActual	Fractional Number	
TXVPressCoolingHiParam	Fractional Number	
TXVPressHeatingLoParam	Fractional Number	
TXVPressHeatingActual	Fractional Number	
TXVPressHeatingHiParam	Fractional Number	
LiquidTempChangeCoolingLoParam	Fractional Number	
LiquidTempChangeCoolingActual	Fractional Number	
LiquidTempChangeCoolingHiParam	Fractional Number	
LiquidTempChangeHeatingLoParam	Fractional Number	
LiquidTempChangeHeatingActual	Fractional Number	
LiquidTempChangeHeatingHiParam	Fractional Number	
CustTag1	String	
CustTag2	String	
CustTag3	String	
CustTag4	String	
CCNINVVersion	Fractional Number	
CCNFlashCode	Fractional Number	
CCNOutdoorCoilTemp	Fractional Number	
CCNOutdoorAirTemp	Fractional Number	
CCNOutdoorSuctionTempHeat	Fractional Number	
CCNSuctionPressHeat	Fractional Number	
CCNLineVoltsHeat	Fractional Number	
CCNOutdoorCoilTempCool	Fractional Number	
CCNOutdoorAirTempCool	Fractional Number	
CCNOutdoorSuctionTempCool	Fractional Number	
CCNSuctionPressCool	Fractional Number	
CCNLineVoltsCool	Fractional Number	
CCNINVSN	String	
CCNProteusOutdoorDischargeTempHeat	Fractional Number	
CCNProteusOutdoorDischargeTempCool	Fractional Number	
CCNProteusINVEEPROMVersion	String	
CCNProteusINVVersion	String	
CCNSerialNo	String	
CCNVersion	String	
CCNErrorMessage	String	
PurgeTime	Integer	

COLUMN NAME	DATA TYPE	KEY TYPE
EqualizationTime	Integer	
RTVersion	Fractional Number	
AC_HP	Fractional Number	
UnitType	Fractional Number	

The complete table is displayed here for effect. Recall that, even if no value is entered for one of these measurement results, space is reserved for its value in the database. In most manufacturing testing, the failure of certain tests will preclude the execution of subsequent tests, hence there will generally be rows in this table that do not have values for all of the columns, resulting in wasted space.

If a new test is to be added, the appropriate parameter columns would be added to this table. All rows in the table for tests that have already been run would, again, have space reserved in the database for these new columns. But values will never be entered into these new columns, hence, space is again wasted. Further, in the interest of preserving historical data, the removal of a test will not result in the removal of any data from the table. Subsequent tests will not be putting values in the columns corresponding to the removed tests, so space is wasted.

It is worth noting that all of the column names in all of the above tables are quite normal and easily read, particularly to someone with experience in a manufacturing-test environment. It is hard to imagine someone copyrighting the names of columns or tables in a database, unless the name of a specific product was being used as a column heading. Both forms of such tables are in use in manufacturing-test environments.

The tables used for storing testing details are "structurally" nearly identical, except for the number of variable parameters that are available for each test and inconsequential differences in the names of the columns. As above, both forms of such tables are in use in manufacturing-test environments.

Here is a depiction of the table named Test_Procedures from the database associated with ECIMOS's ECI application:

COLUMN NAME	DATA TYPE	KEY TYPE
TestID	Integer	
TestSeq	Integer	
TestName	String	
Param1	Fractional Number	
Param2	Fractional Number	
Param3	Fractional Number	
Param4	Fractional Number	
SParam	String	
TestDesc	String	
ScrnFldNum	Integer	
MdbFldNum	Integer	
HasHiLo	Integer	
ContinueOnFail	Integer	
Use Formula Param1	Integer	

COLUMN NAME	DATA TYPE	KEY TYPE
Use Formula Param2	Integer	
Use Formula Param3	Integer	
Use Formula Param4	Integer	
Formula 1 Name	String	
Formula 2 Name	String	
Formula 3 Name	String	
Formula 4 Name	String	
HelpRTF	String	
RecID	Integer	Primary Key

Note that this table has three kinds of variable parameters

- Param1, Param2, Param3, and Param4, which can store fractional numbers;
- SParam, which can store a string; and
- Formula 1 Name, Formula 2 Name, Formula 3 Name, and Formula 4 Name, which can store strings.

Multiple kinds of variable parameters provide flexibility, but too many of them can result in wasted space. While conceivable that all of these variable parameters are used by all of the tests, it is more likely that they are not all used by all of the tests; this table likely suffers from wasted-space issues similar to those suffered by the table ECIMOS's ECI application uses for storing results (see above).

Here is a depiction of the table named RT_Test_Script, in which test details are stored for Carrier's RES applicaton:

COLUMN NAME	DATA TYPE	KEY TYPE
Script	String	Primary Key
StepNo	Integer	Primary Key
Action	String	
P1	String	
P2	String	
P3	String	
P4	String	
ReportName	String	
StepDescription	String	

This table has only one kind of variable parameter: P1, P2, P3, and P4, which can store strings. But it should be pointed out that string fields can effectively store more than just strings. Integers and fractional numbers can both be stored in string fields in human-readable form (e.g., 873 and 12.7, respectively), and then converted to their proper numeric form by the application reading the table. In other words, P1 can store a Param1, an SParam, or a Formula 1 Name, depending on just what is needed. It is likely that this table will result in less wasted space than Test_Procedures (depicted above).

The stored procedures for Carrier's RES application solely perform database-related operations such as retrieving rows that match certain conditions and moving data from the tables accessed by the application to tables for longer-term storage of data; the activities performed by these stored procedures are easily discerned by individuals knowledgeable in structured query language (SQL) and transactional SQL (T-SQL). These stored procedures contain no potentially protectable content along the lines of elaborate data analysis algorithms.

CONCERNS RELATED TO THE CONCLUSIONS OF JAMES M. CHENAULT

This section of the report will speak to the individual conclusions presented by James M. Chenault (hereafter, the author) in his "JMC ECIMOS vs Carrier Report (Exhibit A)."

REGARDING CARRIER PLANT INSPECTION CONCLUSIONS

In the interest of minimizing the size of a database table, the use of parameters, the values of which depend on the specific test being performed, is a prudent and common practice; to specify as distinctly named fields all of the parameters used by all of the tests would necessitate either a distinct table for each type of test or a single table with large numbers of unused fields in each row (see above for why this is a problem). Names such as P1, Param1, and Parameter1 are often used for the first such parameters in a table. The author's investigation into this database structure simply revealed that common practices were followed in its construction.

REGARDING CARRIER DATABASE STRUCTURE CONCLUSIONS

The similarities that exist between the ECIMOS and Carrier databases do not necessarily reflect a "descendent" relationship; that they were developed from a common understanding of how such databases should be built is a simpler explanation for the similarities. The Carrier database is "younger" than the ECIMOS database, but that alone does not imply a "descendent" relationship.

That Carrier would continue to develop its database after switching from ECIMOS's ECI application to Carrier's RES application is to be expected for any progressive manufacturing-test effort. And that both databases contain similar field values (i.e., content rather than structure) is the expected result from testing the same products (see "Regarding the Database Query Comparison Conclusions" for more on this topic).

REGARDING RES SOFTWARE CONCLUSIONS

That Carrier's RES application performs the "functions" listed by the author should be of no surprise to a practicing software developer for manufacturing tests; these, and other, functions are common to manufacturing-test software, as is the intended dependency of Carrier's RES application on the database.

In the interest of clarity, the database contains the <u>names</u> of the tests to be performed, along with model-specific values for adjustable parameters related to the tests, and not the details of how the tests are performed.

REGARDING DATABASE QUERY COMPARISON CONCLUSIONS

The simple descriptions of the actions performed by individual test steps – such as "Set hipot to AC mode" – are the domain of neither copyright nor intellectual-property law. These descriptions were likely arrived at in discussions between the ECIMOS and Carrier, and would be understood by engineers and technicians who work with these types of products. Keeping the descriptions the same would also simplify the transition of the test operators from ECIMOS's ECI application to Carrier's RES application.

The author's discussion around the contents of the comments/description fields in the two databases is reminiscent of the so-called "infinite monkeys theorem," which states that enough monkeys, given enough time striking random keys on typewriters (or, for more relevancy to today, on keyboards), would be able to type out the complete works of Shakespeare; a fun reminder of just how immense numbers can be, but of no use, theoretically or practically, to the questions at hand. It makes complete sense that Carrier would choose to largely preserve descriptions already in use and familiar to the test operators and manufacturing and design engineers. And further exploration of the comments/description fields would provide more evidence, but more evidence of a sensible and common strategy having been implemented.

REGARDING FINAL CONCLUSIONS

Again, the commonalities found between the ECIMOS and Carrier databases are typical and reflect standard practices in migrating content from one database structure to another, especially when it comes to descriptive information to be presented to test operators and design and manufacturing engineers. The design of Carrier's RES application is quite typical for performing manufacturing tests, and demonstrates the intended dependency on an associated database.

MINOR CORRECTIONS

On page 4, the author labels the entry UDL Path in the RES INI file as the database path (i.e., the path to the database itself). The UDL Path is actually the path to the Microsoft or Universal Data Link (.udl) file for the database. This file contains the information necessary to connect to the database, including security credentials, the software driver to be used in communicating with the database, and the "location" of the database. Even this "location" is not necessarily a path to the database itself, while the path to the database would be something akin to C:\Program Files (x86)\Microsoft SQL Server\MSSQL10.SQLEXPRESS\MSSQL\DATA\master.mdf.

On page 17, the author mentions that "Actor frameworks are notoriously difficult to troubleshoot." This framework is no more difficult to troubleshoot than any number of other parallel-execution architectures in LabVIEW™ (or in other programming languages, for that matter). The basic troubleshooting principals true to any code written in LabVIEW™ apply equally well to the actor framework.

On page 31, the author states that "a text based [sic] search of this LabVIEW™ code would not find any of the information found in earlier in this report." A text-based search of the LabVIEW™ code will find the names of the pertinent database tables and fields in the Carrier database and the type of data being written to each field in the database.

CONCLUSION

In the opinion of this senior, practicing manufacturing-test engineer and relational-database designer, there is no evidence of copyright or intellectual-property infringement on the part of Carrier in developing its RES application and the associated database. Further, the databases associated with these applications are designed to store similar content, but the schema employed by Carrier is more advanced and efficient than that used by ECIMOS, particularly from the perspective of relational-database design.

The right to amend or supplement this report, should new information become available, is reserved.

CREDENTIALS

Douglas A. Femec has served as a software and hardware test engineer and architect for the past twenty years, preceded by five years of relational-database design and implementation in laboratory settings. He has earned distinctions in software development as both a Certified LabVIEW™ Architect and a Certified TestStand Developer (awarded by National Instruments™) and in instruction focusing on the effective use of these software products. Except for internal reports and test documentation, test engineers typically do not generate publications and, hence, there are none to list pertinent to this report.

This is the first time Douglas A. Femec has served as an expert witness; commensurate with that level of experience, he is being compensated at the rate of $200/hour by Carrier Corporation.

Douglas A. Femec Signature Oct. 30, 2017 Date

534 High Street, Victor, NY; (585) 924-7737; doug@femec.net

Douglas A. Femec

Experience

2014-2017 Bosch Security Systems Fairport, NY

Test Automation Architect

- Architectural lead for R&D measurements and manufacturing testing of sensor/detector products and for regression testing of controls/software products, both in the Fairport R&D facility and in the Zhuhai manufacturing facility.
- Architected and implemented automated testing using National Instruments' LabVIEW and TestStand, SmartBear's TestComplete, and Cal-Bay's (now Averna's) iVVivi; data analysis and reporting using IntraStage and SQL Server; and the transition of the manufacturing test executive from iVVivi to TestStand.
- Mentored colleagues in both facilities in the software products being used and their integration (particularly with issue-tracking systems such as TargetProcess), data mining (especially statistical process control), and automated testing methods.
- Designed and constructed self-contained security-system environments for end-to-end regression testing, including controllable failure modes and FPGA-based emulation of large numbers of various control modules.

2011-2014 Ventraq, Inc. Pittsford, NY

Senior Quality Software Engineer

- Quality Assurance lead for business analytics product (BIAS) that interfaces with MPPs.
- Developed and executed manual test cases for basic BIAS functionality and customer installations.
- Architected and implemented automated testing using SmartBear's TestComplete and Microsoft's CodedUI, as well as BIAS itself.
- Developed and deployed coordinated, JIRA-driven automated testing across multiple virtual machines (VMs), including data mining.
- Principal administrator for corporate JIRA, Confluence, Splunk, and TestComplete installations.
- Coordinated testing efforts across worldwide internal and customer facilities.
- Mentored colleagues in manual and automated testing methods.
- Grew experience in Visual C#, .NET, IBM Streams, Ruby-on-Rails, Java script, and in SQL Server, IBM/Netezza, Teradata, and Oracle database products.
- Transitioned issue tracking from VersionOne to JIRA.
- Provided internal training on new releases of BIAS and on new internal tools (JIRA, TestComplete, CodedUI, and Splunk).
- Earned Scrum Master certification.

534 High Street, Victor, NY; (585) 924-7737; doug@femec.net

Douglas A. Femec

2004-2011 MKS Instruments, Inc. Rochester, NY

Lead Software Test Engineer

- Architected and developed multiple LabVIEW and TestStand applications for manufacturing testing and calibration of high-power RF and DC generators for semiconductor manufacturing and medical imaging, resulting in noteworthy increases in testing throughput, test coverage and consistency, and data-capture accuracy.
- Work utilized LabVIEW, TestStand, and Visual Basic (6 and .NET) programming languages for interfacing with serial- and GPIB-based instrumentation and plug-in data-acquisition cards and for automated deployment and upgrading of developed applications.
- Chaired emergency medical response unit and teams developing employee rewards and recognition programs and facility-wide reuse and recycling efforts.
- Mentored colleagues in how to develop code well and how to think about manufacturing testing as distinct from design testing.
- Developed course materials and lectures for and taught corporate-specific LabVIEW and custom-application courses.
- Trained for and assisted in conducting internal ISO and other related regulatory audits.
- Provided voice talent for internal presentations.

1996-2004 Viewpoint Systems Rochester, NY

Systems/Software Engineer

- Designed and developed cost-effective and scalable custom engineering software applications and test systems for a diverse set of customers.
- Applications involved intensive data acquisition and control, signal processing, statistical analysis, and mathematical programming in LabVIEW, TestStand, SQL, Visual Basic, and Fortran.
- Responsible for developing and managing a one-stop-shopping offering for motion applications, including integration of software with mechanical and control hardware for varied uses.
- Respected and experienced National Instruments Certified Professional Instructor in LabVIEW.
- Managed training operations, putting the necessary pieces in place to allow the position to be handed off.
- Developed course materials and lectures for and taught client-specific LabVIEW and custom-application courses.
- Earned LabVIEW Architect and TestStand Developer certifications.

534 High Street, Victor, NY; (585) 924-7737; doug@femec.net

Douglas A. Femec

1989-1996 Idaho National Laboratory Idaho Falls, ID

Senior Scientist

- Algorithm, software, and hardware research and development related to the computer-controlled acquisition, analysis, and reporting of radiation measurement data for medical-isotope, nuclear-waste, and metallurgical characterization.
- Relational-database and associated user-interface development related to sample tracking, quality control, and fiscal management.
- Synthesis of new phosphazene prepolymers for structural and gas-separation membrane applications.
- Coatings and modifiers of fiber optic surfaces for desired analyte binding in evanescent-wave chemical sensors.
- Analytical methodologies for Fourier-transform infrared spectroscopy of solids and powders.
- Design of apparatus for chemical vapor deposition.
- Developed proposals for lithium hydride slurry-fueled ion-propulsion systems.
- Initiated and directed a seminar series on advanced instrumentation available at the INEL.
- Held a Department of Energy “Q” clearance in association with analytical-methods development efforts.

Education

1982-1986 Cornell University Ithaca, NY

Ph.D., Physical Inorganic Chemistry

- Synthesized, characterized, and conducted dynamic NMR, crystallographic, and molecular orbital studies of new titanium group metallocene and metal chelate compounds.
- Developed and maintained research group's crystallographic and NMR line-shape analysis programs and data archiving.

1980-1982 Cornell University Ithaca, NY

M.S., Inorganic Chemistry

- GPA of 3.5 out of 4.0

1976-1980 University of Kansas Lawrence, KS

B.S., Chemistry; B.A., Biochemistry

- Undergraduate Research Assistant performing calculations for (1) correlating isotope effects on vibrational frequencies to transition-state structures of enzymic methyl-transfer reactions and (2) determining gas-phase transition-state structures for the addition of water across carbonyl bonds, including the ability of hydrating water molecules to stabilize transition-state structures.
- Summerfield Scholar, GPA of 3.9 out of 4.0